Assume **HourlyEmployee** extends **Employee**. Both classes have a **toString** method.

1. **Employee emp = new HourlyEmployee(…)**

This is an example of ***polymorphic substitution*** because we have the **Employee** reference **emp** referencing an **HourlyEmployee** object.

**Definition:**  ***Polymorphic substitution*** refers to the is-a relationship between superclass and subclass that allows us to use a subclass object anywhere a superclass object would be allowed – this means that a reference variable for the supertype may actually store a reference to an instance of the subclass.

1. Suppose we send the **toString** message to **emp**. At runtime, the runtime system determines that **emp** actually references an **HourlyEmployee**.

This is an example of ***late-binding polymorphism*** because the runtime system is overriding the **toString** method from the **Employee** class (which is the declared type of **emp**) and is determining ***at runtime*** which method to invoke based on the actual type that **emp** is referencing.

**Definition:**  The ability to override methods, coupled with the run-time determination of which method to invoke is called ***late-binding polymorphism***.

1. Suppose **HourlyEmployee** and **Employee** do ***not*** have a **toString** method. If we send a **toString** method to **emp**, Java will search the class hierarchy until it finds a **toString** method it can use. In this case, it will find a **toString** method in the **Object** class and will invoke that method.

This is an example of ***inheritance***, not polymorphism. There is no overriding method, so Java attempts to find an inherited method.

**Definition:**  Subclasses extend the behavior and state of the parent class. The subclass can have additional attributes and methods. However, the attributes and methods of the parent class are ***inherited*** by all subclasses of the parent class.